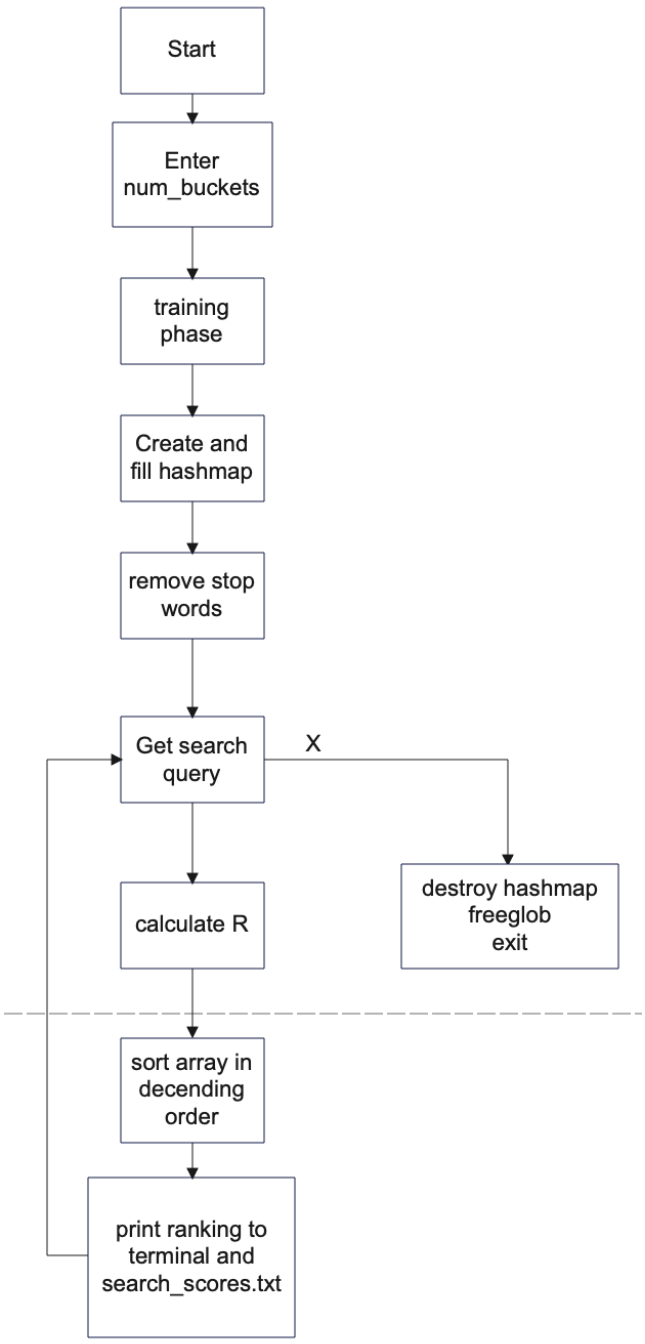
Seeam Khan
12/10/22

Project 5 Report



The general order is as follows: prompt the user for the number of buckets the hashmap will have, fill the hashmap with all words from all the documents using glob, remove all the words that in the hashmap that appear in every document, get the search query from user, loop through every word in the search query and get the ranking for every document, sort the rankings in descending order, print ranking to terminal and store scores in search_scores.txt.

I implemented the hashmap with lists of lists, where every word node has a linked list of documents that contained that word. I chose to implement this instead of keeping the simple hashmap since I concluded it would be much easier and faster when getting the document frequency, term frequency, and removing stop words. The word node contained: char * word that contained the word, an int value that stored the number of documents that contained that word, a pointer to the next word node in that bucket, and a pointer to the list of documents that contained that word. The document nodes contained: a char* document_id that held the name of the document, an integer value that stored the number of times the respective word appeared in the document, and a pointer to the

next document node in that list. From homework 6 I used the hash, put, get, create, remove, and destroy. I altered these functions to suit the lists of lists hashmap by mostly adding another loop to go through the list of docs after the correct word node is found.

Functions:
int main(void):

- In main the program prompts the user to enter the number of buckets the hashmap will contain, this number is stored in a num_buckets variable. After this a glob_t struct is created and "p5docs/*.txt" is passed into it. The program then passes the glob_t struct and numbuckets into training where the hashmap is created. After the hashmap is created, we go into a while loop that keeps looping until the user enters 'X', where the program calls hm_destroy() and freeglob(). If the user enters a search query, the program goes into read_query() where the scores are calculated. After this main calls append() which fills search_scores.txt with the documents and their scores. Then right before the while loop ends, the program resets the array of scores to 0 for the next iteration.

struct hashmap* hm_create(int num_buckets);

- hm_create takes in the number of buckets the user has entered, it callocs 1 size of struct hashmap, and sets its num_elements to 0 and its num_buckets to num_buckets. It also callocs num_buckets sizeof struct llnode and points maps to it. It returns the hashmap that was just created.

int hm_get_tf(struct hashmap* hm, char* word, char* document_id);

- This function returns the term frequency of a certain word in a certain document. It calculates the hash and iterates through that bucket till it finds the right word, then iterates through the doc list until the right document is found, and returns the tf of that lldoc node.
- If a tf is not found it returns -1.

Seeam Khan
12/10/22

`int hm_get_df(struct hashmap* hm, char* word);`

- This function returns the doc frequency of a certain word. It calculates the hash and iterates through that bucket till it finds the right word, then returns the df of that llnode.
- If a df is not found, it returns -1.

`void hash_table_insert(struct hashmap* hm, char* word, char* document_id, int num_occurrences);`

- This function puts a word document pair in the hashmap. It first calls hash() and goes into that bucket. It then searches if that word node exists, if it doesn't a new node is added to the end of that bucket list, and a new lldoc node is added to that word's docs list. If the word node exists it iterates through the list of docs and searches for the lldoc node. If it exists the term frequency is updated, if it isn’t found, a new lldoc node is created and put at the end of that word’s document list.

`void hm_remove(struct hashmap* hm, char* word);`

- This function takes in a hashmap pointer and word and returns nothing. The function calls hash() and iterates that bucket till the word node is found. It then iterates the word doc list freeing every lldoc node in that list, then finally freeing the word node and redirecting the pointers.

`void hm_destroy(struct hashmap* hm);`

- This function iterates every bucket of the passed in hm and calls hm_remove() on every word node before finally freeing the hashmap itself.

`int hash(struct hashmap* hm, char* word);`

- This calculates the bucket a word should or would be placed in. it's calculated by getting the sum of every letter in the word and modding that by the number of buckets in hm.

Seeam Khan
12/10/22

```
void printMap(struct hashmap* hm);
```
- This function was for debugging purposes. It iterated through every bucket in the hashmap and printed the word, its df, and every doc in that word's doc list along with their tf.

```
struct hashmap* training(glob_t glob, int buckets);
```
- This function took in the glob_t struct and the number of buckets. It used the number of buckets to call hm_create and create a new hashmap. It then iterated every doc in glob.gl_pathv and got every word in that doc, and called hash_table_insert() on that word and document.
- The function then calls stop_word() to remove all the stop words in the hashmap.

```
void read_query(char * search, struct hashmap* hm, glob_t glob,
double score[]);
```
- This function takes in the search query, a hashmap, a glob_t struct, and the array of scores.
- It parses every word in the search query and calls rank on it with every document. The rank is then stored in the matching index the document_id is stored in glob.gl_patchv.
- It then calls sort on the score array and glob struct.

```
void sort(double score[], glob_t glob);
```
- This function takes in the array of scores and a glob struct. It sorts the score array from greatest to lowest using bubble sort. For any change made to the score array, the same change is made in the array of path names in glob. This is done to keep all the indexes matching the right score and document.

```
void stop_word(struct hashmap* hm, glob_t glob);
```
- This function iterates through every bucket in the hashmap, and if any word’s df is equal to the number of documents (glob.gl_pathc), hm_remove is called on that word.

Seeam Khan
12/10/22

```
double rank(char * word, struct hashmap* hm, char* document_id,
glob_t glob);
```
- This function calls hm_get_df and hm_get_tf and calculates the tf-idf score and returns it.

```
void append(double scores[], glob_t glob, char * destination);
```
- This function opens search_scores.txt and appends to it. It gets the doc names from glob and scores from the scores array. The doc names in glob are “p5docs/*.txt” so I had to create another string that is a substring of the doc name from glob containing only the last 6 characters, this substring is appended to seach_scores.txt and is printed in the terminal after the user enters their search query.

```
void swap_int(double* xp, double* yp);
```
- Swapped values.

```
void swap1(char **str1_ptr, char **str2_ptr);
```
- Swapped char pointers.